

**厦门大学数据结构实验报告**

**2017-2018学年第一学期**

**实验名称：实验五 全国交通咨询模拟**

**系 别： 智能系**

**学生学号： 22920162204046**

**学生姓名： 王凯**

题目： 全国交通咨询模拟

1. **需求分析**

【基本要求】

(1)对以下6种常用的内部排序算法进行比较:起泡排序、直接插入排序、简单选择排序、快速排序、希尔排序、堆排序

(2)待排序表的表长不小于100;其中的数据要用伪随机数产生程序产生;至少要用5组不同的输入数据作比较;比较的指标为有关键字参加的比较次数和关键字的移动次数(关键字交换计为3次移动)。

(3)最后要对结果作出简单分析,包括对各组数据得出结果波动大小的解释。

**二、概要设计**

**1：抽象数据类型**

**typedef** int Status**;**

#define OK 1

#define ERROR 0

#define RANGE 10000

#define ARRAY 100

int Start**();**//界面

Status output**(**int a**[**ARRAY**]);**//打印函数

void BubbleSort**(**int a**[**ARRAY**]);** //起泡排序

Status insertSort**(**int a**[**ARRAY**]);** //直接插入排序

Status SelectSort**(**int a**[**ARRAY**]);** //简单选择排序

Status Partition**(**int a**[**ARRAY**],** int low**,** int high**,** int **&**s**,** int **&**l**);**//快速排序

void Qsort**(**int a**[**ARRAY**],** int low**,** int high**,** int **&**s**,** int **&**l**);**

void QuickSort**(**int a**[**ARRAY**]);**

void ShellInsert**(**int a**[**ARRAY**],** int d**,** int **&**s**,** int **&**l**);**

void ShellSort**(**int a**[**ARRAY**]);** //希尔排序

void HeapAdjust**(**int a**[**ARRAY**],**int i**,**int A**,** int **&**s**,** int **&**l**);**

void HeapSort**(**int a**[**ARRAY**]);** //堆排序

void BInsertSort**(**int a**[**ARRAY**]);** //折半插入排序

void BiInsert\_Sort**(**int a**[**ARRAY**]);** //二路插入

void merge**(**int X**[],**int Z**[],**int s**,**int u**,**int v**);**//将有序的X[s..u]和X[u+1..v]归并为有序的Z[s..v]

void mergePass**(**int X**[],**int Y**[],**int n**,**int t**);**

void mergeSort**(**int a**[**ARRAY**]);**//二路归并排序

int GetNumInPos**(**int num**,**int pos**);**// 找到num的从低到高的第pos位的数据

void RadixSort**(**int a**[**ARRAY**]);**//基数排序

**2：主程序**

int main(){

while(1){

接受命令；

处理命令；

**}**

**}**

1. **详细设计**

**1：存储结构**

以数组存储数据

**2、详细代码展示**

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

**using** **namespace** std**;**

#include"InternalRanking.h"

int Start**()**

**{**

printf**(**" \*-----------此系统提示说明-------------\*\n"**);**

printf**(**" \*请输入你要进行的步骤的序号: \*\n"**);**

printf**(**" \* 1:起泡排序 \*\n"**);**

printf**(**" \* 2:直接插入排序 \*\n"**);**

printf**(**" \* 3:简单选择排序 \*\n"**);**

printf**(**" \* 4:快速排序 \*\n"**);**

printf**(**" \* 5:希尔排序 \*\n"**);**

printf**(**" \* 6:堆排序 \*\n"**);**

printf**(**" \* 7:折半插入排序 \*\n"**);**

printf**(**" \* 8:二路插入排序 \*\n"**);**

printf**(**" \* 9:归并排序 \*\n"**);**

printf**(**" \* 10:基数排序 \*\n"**);**

printf**(**" \* 11:一次性输出所有排序结果 \*\n"**);**

printf**(**" \* 0:退出系统 \*\n"**);**

printf**(**" \*--------------------------------------\*\n"**);**

**return** 1**;**

**}**

Status output**(**int a**[**ARRAY**])**

//打印函数

**{**

int i**,** k**=**0**;**

**for(**i**=**0**;** i**<**ARRAY**;** i**++)**

**{**

printf**(**"%6d"**,** a**[**i**]);**

k**++;**

**if(**k**%**10**==**0**)**

**{**

printf**(**"\n"**);**

**}**

**}**

**return** OK**;**

**}**

void BubbleSort**(**int a**[**ARRAY**])**

**{**//起泡排序

int i**,** j**,** t**,** s**=**0**,** l**=**0**;**

**for(**i**=**0**;** i**<**ARRAY**;** i**++)**

**{**

**for(**j**=**0**;** j**<(**ARRAY**-**i**);** j**++)**

**{**

**if(**a**[**j**]>**a**[**j**+**1**])**

**{**

t **=** a**[**j**];**

a**[**j**]** **=** a**[**j**+**1**];**

a**[**j**+**1**]** **=** t**;**

l**++;**

**}**

s**++;**

**}**

**}**

printf**(**"起泡排序：\n"**);**

output**(**a**);**

printf**(**"其比较次数和移动次数： %d %d\n"**,** s**,** l**);**

**}**

Status insertSort**(**int a**[**ARRAY**])**

**{**//直接插入排序

int i**,** j**,** k**,** s**=**0**,** l**=**0**;** //k作为辅助空间；

**for(**i**=**1**;** i**<**ARRAY**;** **++**i**)**

**{**

**if(**a**[**i**]<**a**[**i**-**1**])**

**{**

k **=** a**[**i**];**

s**++;**

**for(**j **=** i**-**1**;** j**>=**0**&&**a**[**j**]>**k**;** **--**j**)**

**{**

a**[**j**+**1**]** **=** a**[**j**];**

l**++;**

**}**

a**[**j**+**1**]** **=** k**;**

l**++;**

**}**

**}**

printf**(**"直接插入排序：\n"**);**

output**(**a**);**

printf**(**"比较次数和移动次数： %d %d\n"**,** s**,** l**);**

**return** OK**;**

**}**

Status Partition**(**int a**[**ARRAY**],** int low**,** int high**,** int **&**s**,** int **&**l**)**

**{**//快速排序

int k**;**

k **=** a**[**low**];**

**while(**low **<** high**)**

**{**

**while(**low**<**high**&&**a**[**high**]>=**k**)**

**{**

high **--;**

s**++;**

**}**

a**[**low**]** **=** a**[**high**];**

**while(**low**<**high **&&** a**[**low**]<=**k**)**

**{**

low **++;**

s**++;**

**}**

a**[**high**]** **=** a**[**low**];**

s**++;**

**}**

a**[**low**]** **=** k**;**

l**++;**

**return** low**;**

**}**

void Qsort**(**int a**[**ARRAY**],** int low**,** int high**,** int **&**s**,** int **&**l**)**

**{**

int pivoloc**;**

**if(**low**<**high**)**

**{**

pivoloc **=** Partition**(**a**,** low**,** high**,** s**,** l**);**

Qsort**(**a**,** low**,** pivoloc**-**1**,** s**,** l**);**

Qsort**(**a**,** pivoloc**+**1**,** high**,** s**,** l**);**

**}**

**}**

void QuickSort**(**int a**[**ARRAY**])**

**{**

int s**=**0**,** l**=**0**;**

Qsort**(**a**,** 0**,** 99**,** s**,** l**);**

printf**(**"快速排序：\n"**);**

output**(**a**);**

printf**(**"比较次数和移动次数： %d %d\n"**,** s**,** l**);**

**}**

Status SelectSort**(**int a**[**ARRAY**])**

**{**//简单选择排序

int i**,** j**,** k**,** l**=**0**,** s**=**0**,** t**,** u**;**

**for(**i**=**0**;** i**<**ARRAY**;** i**++)**

**{**

k**=**a**[**i**];**

u**=**i**;**

**for(**j**=**i**;** j**<**ARRAY**;** j**++)**

**{**

s**++;**

**if(**a**[**j**]<**k**)**

**{**

k **=** a**[**j**];**

u **=** j**;**

**}**

**}**

s**++;**

**if(**i**!=**u**)**

**{**

t**=**a**[**i**];**

a**[**i**]** **=** a**[**u**];**

a**[**u**]** **=** t**;**

l**++;**

**}**

**}**

printf**(**"简单选择排序：\n"**);**

output**(**a**);**

printf**(**"比较次数和移动次数： %d %d\n"**,** s**,** l**);**

**return** OK**;**

**}**

void ShellInsert**(**int a**[**ARRAY**],** int d**,** int **&**s**,** int **&**l**)**

**{**

int i**,** j**,** k**;**

**for(**i **=** d**;** i**<**ARRAY**;** **++**i**)**

**{**

**if(**a**[**i**]<**a**[**i**-**d**])**

**{**

s**++;**

k **=** a**[**i**];**

**for(**j**=**i**-**d**;** j**>=**0**&&**k**<**a**[**j**];** j**-=**d**)**

**{**

s**++;**

a**[**j**+**d**]** **=** a**[**j**];**

l**++;**

**}**

a**[**j**+**d**]** **=** k**;**

l**++;**

**}**

**}**

**}**

void ShellSort**(**int a**[**ARRAY**])**

**{**//希尔排序

int t**=**4**,** i**,** s**=**0**,** l**=**0**;**

int dlta**[]** **=** **{**40**,** 13**,** 4**,** 1**};**

**for(**i**=**0**;** i**<**t**;** i**++)**

**{**

ShellInsert**(**a**,** dlta**[**i**],** s**,** l**);**

**}**

printf**(**"希尔排序：\n"**);**

output**(**a**);**

printf**(**"比较次数和移动次数： %d %d\n"**,** s**,** l**);**

**}**

void HeapAdjust**(**int a**[**ARRAY**],**int i**,**int A**,** int **&**s**,** int **&**l**)**

**{**

int r**,**j**;**

r **=** a**[**i**];**

**for(**j**=**2**\***i**;** j**<**A**;** j**\*=**2**)**

**{**

**if(**j**<**A**&&**a**[**j**]<**a**[**j**+**1**])** **++**j**;**

s**++;**

**if(!(**r**<**a**[**j**]))** **break;**

s**++;**

a**[**i**]** **=** a**[**j**];**

l**++;**

i **=** j**;**

**}**

a**[**i**]** **=** r**;**

**}**

void HeapSort**(**int a**[**ARRAY**])**

**{**//堆排序

int i**,** t**,** s**=**0**,** l**=**0**;**

**for(**i**=**ARRAY**/**2**;** i**>=**0**;** **--**i**)**

**{**

HeapAdjust**(**a**,** i**,** ARRAY**,** s**,** l**);**

**}**

**for(**i**=**ARRAY**-**1**;** i**>**0**;** **--**i**)**

**{**

t **=** a**[**0**];**

a**[**0**]** **=** a**[**i**];**

a**[**i**]** **=** t**;**

l**++;**

HeapAdjust**(**a**,** 0**,** i**-**1**,** s**,** l**);**

**}**

printf**(**"堆排序：\n"**);**

output**(**a**);**

printf**(**"其比较次数和移动次数： %d %d\n"**,** s**,** l**);**

**}**

void BInsertSort**(**int a**[**ARRAY**])**

//折半插入排序

**{**

int i**,** j**,** k**,** low**,** high**,** m**,**s**=**0**,**l**=**0**;**

**for(**i **=** 1**;** i **<**ARRAY**;** i**++)** **{**

low **=** 0**;**

high **=** i **-** 1**;**

**while(**low **<=** high**)** **{**

m **=** **(**low **+** high**)** **/** 2**;**

**if(**a**[**m**]** **>** a**[**i**])**

high **=** m **-** 1**;**

**else**

low **=** m **+** 1**;**

s**++;**

**}**

**if(**j**!=**i**-**1**){**

int temp **=** a**[**i**];**

**for(**k **=** i **-** 1**;** k **>=** high **+** 1**;** k**--)**

**{**

a**[**k **+** 1**]** **=** a**[**k**];**

l**++;**

**}**

a**[**k **+** 1**]** **=** temp**;**

l**++;**

**}**

**}**

printf**(**"折半插入排序：\n"**);**

output**(**a**);**

printf**(**"其比较次数和移动次数： %d %d\n"**,** s**,** l**);**

**}**

void BiInsert\_Sort**(**int a**[**ARRAY**])**

//二路插入排序的算法

**{**

int i**,** first**,** final**,** k**,**s**=**0**,**l**=**0**;**

int temp**[**ARRAY**];**

first **=** final **=** 0**;**

temp**[**0**]** **=** a**[**0**];**

**for** **(**i **=** 1**;** i**<**ARRAY**;**i **++)**

**{**

**if** **(**a**[**i**]** **<** temp**[**first**])**

**{** // 待插入元素比最小的元素小

first **=** **(**first **-** 1 **+** ARRAY**)** **%** ARRAY**;**

temp**[**first**]** **=** a**[**i**];**

l**++;**

s**++;**

**}** **else** **if** **(**a**[**i**]** **>** temp**[**final**])**

**{** // 待插入元素比最大元素大

final **=** **(**final **+** 1 **+** ARRAY**)** **%**ARRAY**;**

temp**[**final**]** **=** a**[**i**];**

l**++;**

s**++;**

**}** **else** **{** // 插入元素比最小大，比最大小

k **=** **(**final **+** 1 **+** ARRAY**)** **%** ARRAY**;**

**while** **(**temp**[((**k **-** 1**)** **+** ARRAY**)** **%** ARRAY**]** **>** a**[**i**])**

**{**

temp**[(**k **+** ARRAY**)** **%** ARRAY**]** **=**temp**[(**k **-** 1 **+** ARRAY**)** **%** ARRAY**];**

k **=** **(**k **-** 1 **+** ARRAY**)** **%** ARRAY**;**

s**++;**

**}**

temp**[(**k **+** ARRAY**)** **%** ARRAY**]** **=** a**[**i**];**

final **=** **(**final **+** 1 **+** ARRAY**)** **%** ARRAY**;**

l**++;**

s**++;**

**}**

**}**

// 将排序记录复制到原来的顺序表里

**for** **(**k **=** 0**;** k **<**ARRAY**;** k **++){**

a**[**k**]** **=** temp**[(**first **+** k**)** **%** ARRAY**];**

l**++;**

**}**

printf**(**"二路插入排序：\n"**);**

output**(**a**);**

printf**(**"其比较次数和移动次数： %d %d\n"**,** s**,** l**);**

**}**

void merge**(**int X**[],** int Z**[],** int s**,** int u**,** int v**,**int **\***m**,**int **\***n**)**

//将有序的X[s..u]和X[u+1..v]归并为有序的Z[s..v]

**{**

int i**,** j**,** q**;**

i **=** s**;**

j **=** u **+** 1**;**

q **=** s**;**

**while(** i **<=** u **&&** j**<=** v **)**

**{**

**if(** X**[**i**]** **<=** X**[**j**]** **){**

Z**[**q**++]** **=** X**[**i**++];**

**(\***m**)++;**

**(\***n**)++;**

**}**

**else{**

Z**[**q**++]** **=** X**[**j**++];**

**(\***m**)++;**

**(\***n**)++;**

**}**

**}**

**while(** i **<=** u **){** //将X中剩余元素X[i..u]复制到Z

Z**[**q**++]** **=** X**[**i**++];**

**(\***n**)++;**

**}**

**while(** j **<=** v **)** **{** //将X中剩余元素X[j..v]复制到Z

Z**[**q**++]** **=** X**[**j**++];**

**(\***n**)++;**

**}**

**}**

void mergePass**(**int X**[],**int Y**[],**int n**,**int t**,**int **\***s**,**int **\***l**)**

**{**

int i **=** 0**,** j**;**

**while(** n **-** i **>=** 2 **\*** t **)** //将相邻的两个长度为t的各自有序的子序列合并成一个长度为2t的子序列

**{**

merge**(**X**,** Y**,** i**,** i **+** t **-** 1**,** i **+** 2 **\*** t **-** 1**,**s**,**l**);**

i **=** i **+** 2 **\*** t**;**

**}**

**if(** n **-** i **>** t **)** //若最后剩下的元素个数大于一个子序列的长度t时

merge**(**X**,** Y**,** i**,** i **+** t **-** 1**,** n **-** 1**,**s**,**l**);**

**else** //n-i <= t时，相当于只是把X[i..n-1]序列中的数据赋值给Y[i..n-1]

**for(** j **=** i **;** j **<** n **;** **++**j **)**

**{**

Y**[**j**]** **=** X**[**j**];**

**(\***l**)++;**

**}**

**}**

void mergeSort**(**int a**[**ARRAY**])**

//二路归并排序

**{**

int i**=**1**,**s**=**0**,**l**=**0**;**

int **\***b**=** **(**int **\*)**malloc**(sizeof(**int**)** **\***ARRAY**);**

**while(**i**<**ARRAY**)**

**{**

mergePass**(**a**,** b**,** ARRAY**,** i**,&**s**,&**l**);**

i**\*=** 2**;**

mergePass**(**b**,** a**,** ARRAY**,** i**,&**s**,&**l**);**

i**\*=** 2**;**

**}**

free**(**b**);**

printf**(**"二路归并排序：\n"**);**

output**(**a**);**

printf**(**"其比较次数和移动次数： %d %d\n"**,**s**,**l**);**

**}**

int GetNumInPos**(**int num**,**int pos**)**

// 找到num的从低到高的第pos位的数据

**{**

int temp **=** 1**;**

**for** **(**int i**=**0**;** i**<**pos**-**1**;** i**++)**

temp**\*=**10**;**

**return** **(**num**/**temp**)%**10**;**

**}**

void RadixSort**(**int a**[**ARRAY**])**

//基数排序

**{**

int i**,**j**,**k**,**pos**,**num**,**index**,**s**=**0**,**l**=**0**;**

int **\***b**[**10**];** //分别为0~9的序列空间

**for** **(**i**=**0**;**i**<**10**;**i**++)**

**{**

b**[**i**]=(**int **\*)**malloc**(sizeof(**int**)\*(**ARRAY**+**1**));**

b**[**i**][**0**]** **=** 0**;** //index为0处记录这组数据的个数

**}**

**for** **(**pos **=** 1**;** pos **<=**31**;** pos**++)** //从个位开始到31位

**{**

**for** **(**i **=** 0**;** i **<**ARRAY**;** i**++)** //分配过程

**{**

num **=** GetNumInPos**(**a**[**i**],** pos**);**

index **=** **++**b**[**num**][**0**];**

b**[**num**][**index**]** **=** a**[**i**];**

l**++;**

**}**

**for** **(**i **=** 0**,**j**=**0**;**i **<**10**;**i**++)** //收集

**{**

**for** **(**k **=** 1**;** k**<=**b**[**i**][**0**];**k**++)**

**{**

a**[**j**++]=**b**[**i**][**k**];**

s**++;**

**}**

b**[**i**][**0**]=**0**;**

**}**

**}**

printf**(**"基数排序：\n"**);**

output**(**a**);**

printf**(**"其比较次数和移动次数： %d %d\n"**,** s**,** l**);**

**}**

1. **调试分析**
2. 此系统实现了10种内部排序（分别演示和对于一组数据，10种排序同时排序比较），
3. 实现了可以多次进行多次排序功能,并且有提醒机制，还可以在适当时候退出系统，界面、人机交互较为友好。
4. 给出了每种内部排序算法排序过程中进行的交换次数、比较次数。
5. 实验数据是通过随机数产生的。
6. 可以发现，快速排序再比较次数、交换次数上都比较少。
7. 算法时间复杂度分析：

1：起泡排序：O（n2）

2:直接插入排序 : O（n2）

3:简单选择排序: O（n2）

4:快速排序 : O(nlogn)

5:希尔排序: O(n1.3)

6:堆排序: O(nlogn)

7:折半插入排序: O（n2）

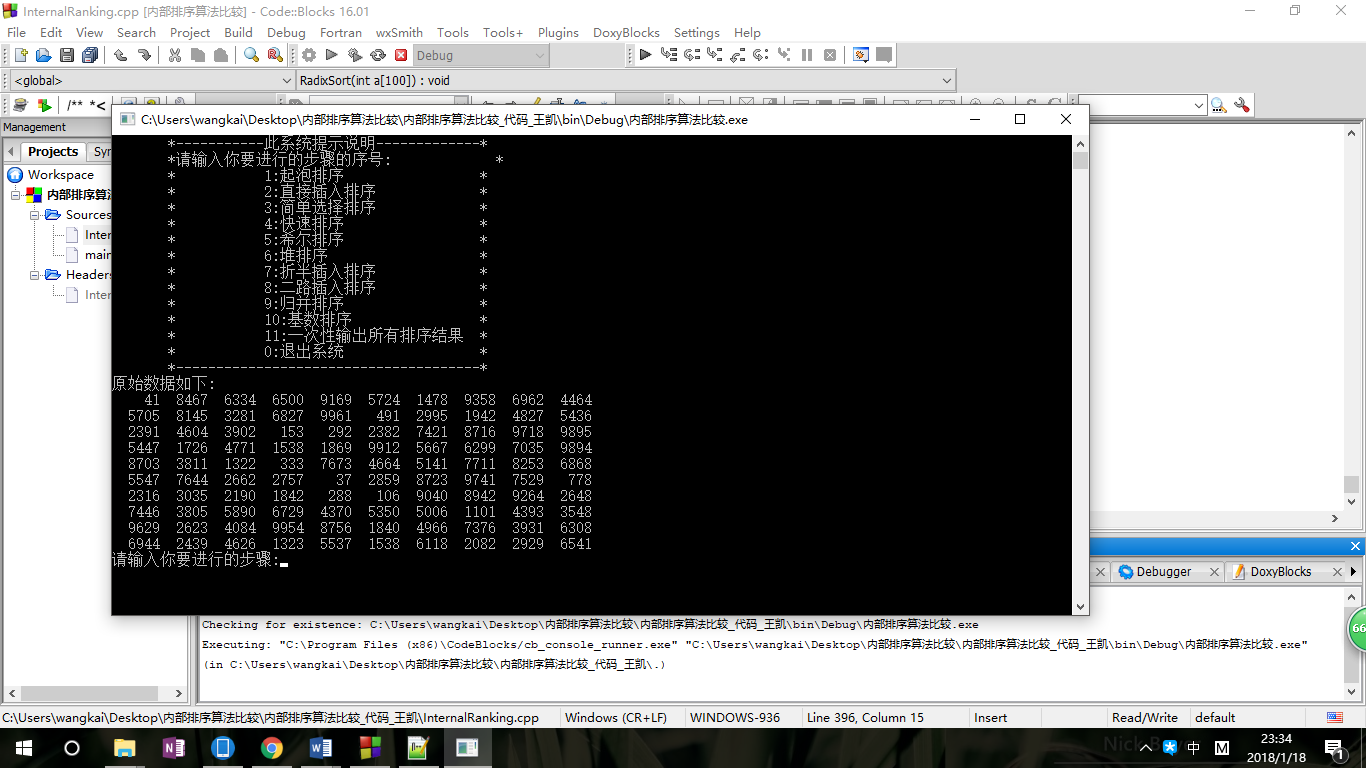
8:二路插入排序 : O（n2）

9:归并排序:O(nlogn)

10:基数排序 : O(d(n+rd))

1. **用户使用说明**

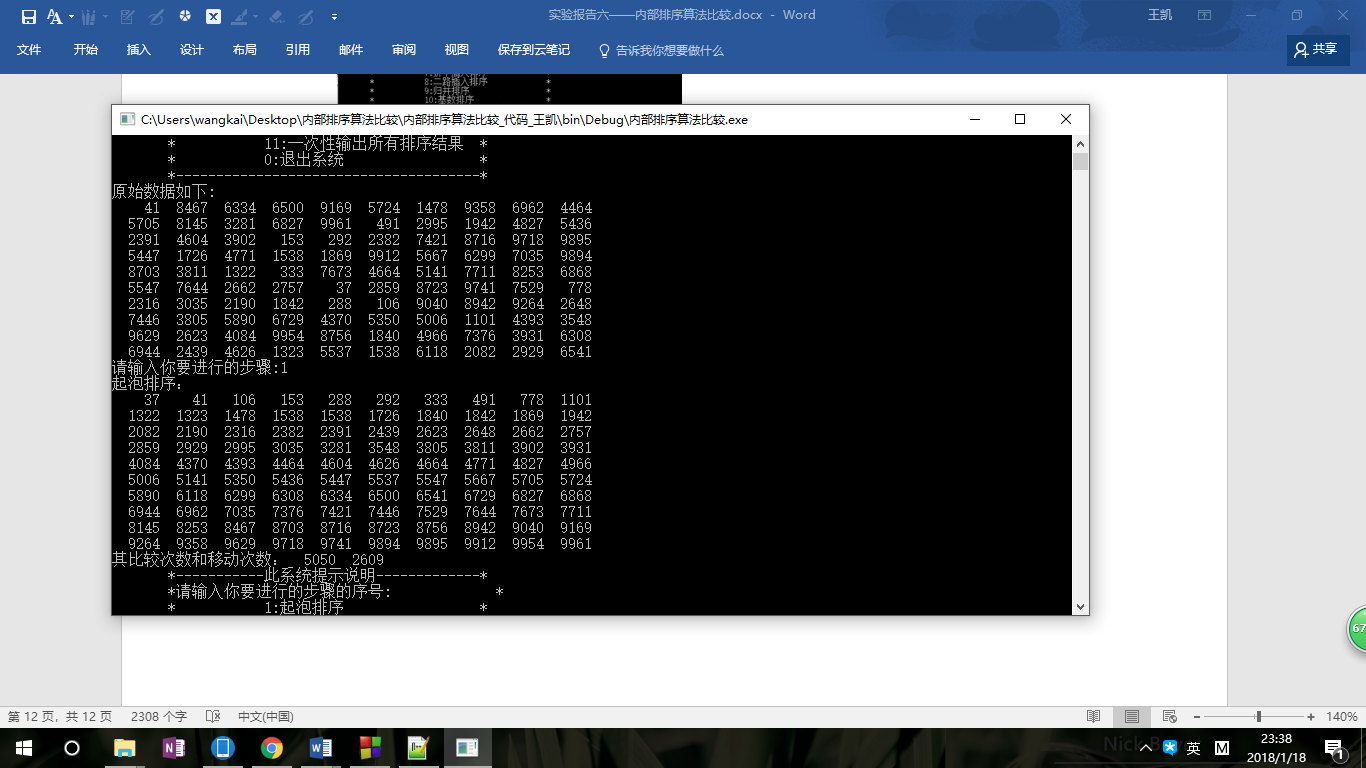
用户只需根据系统的的提醒操作，下面系统菜单界面：



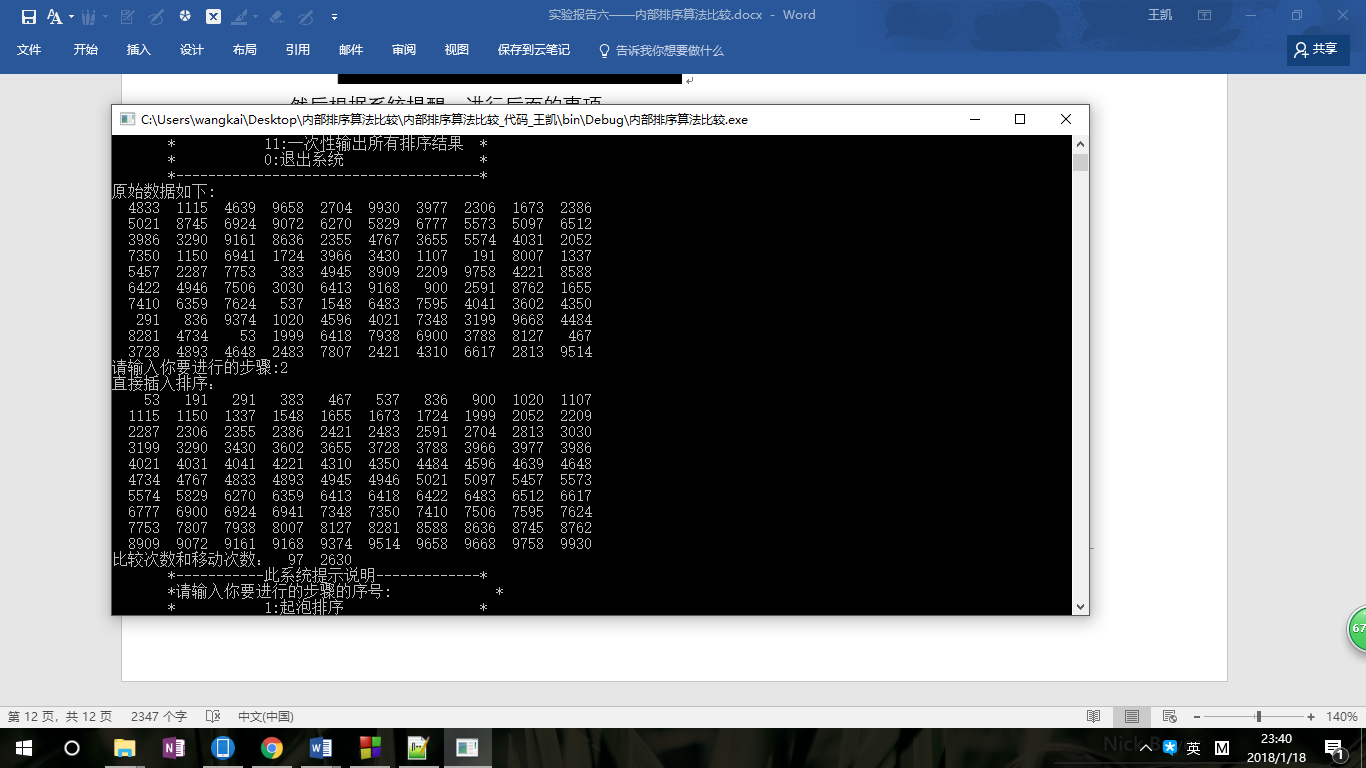
然后根据系统提醒，进行后面的事项。

1. **测试结果**

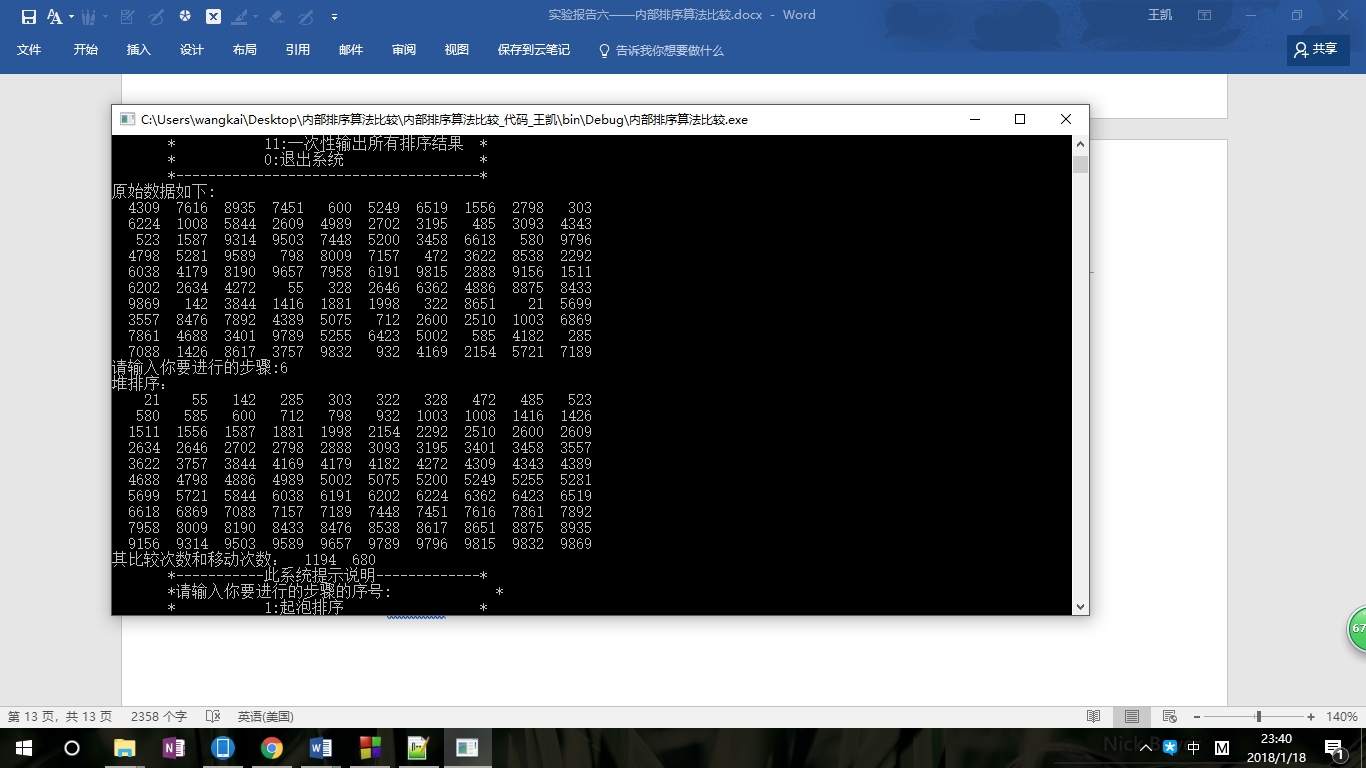
1：起泡排序



2：直接插入排序

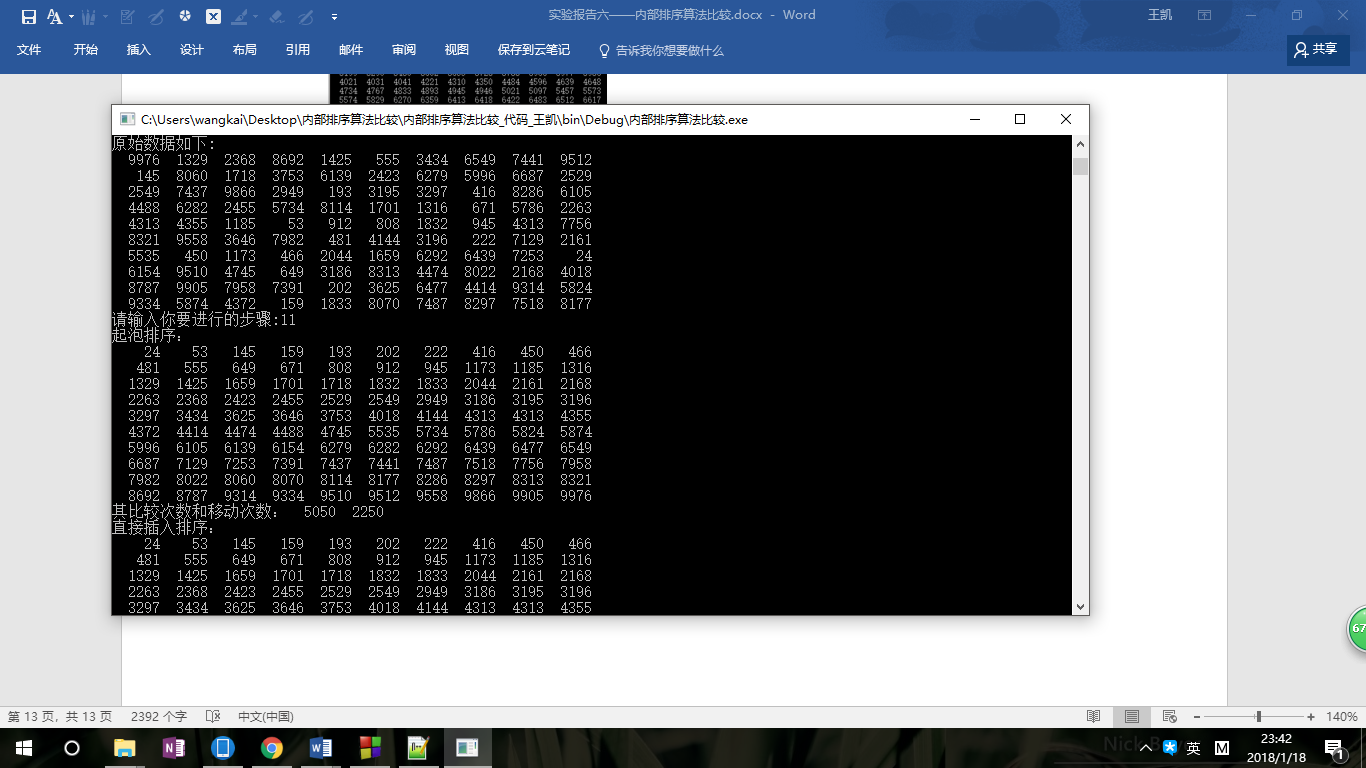
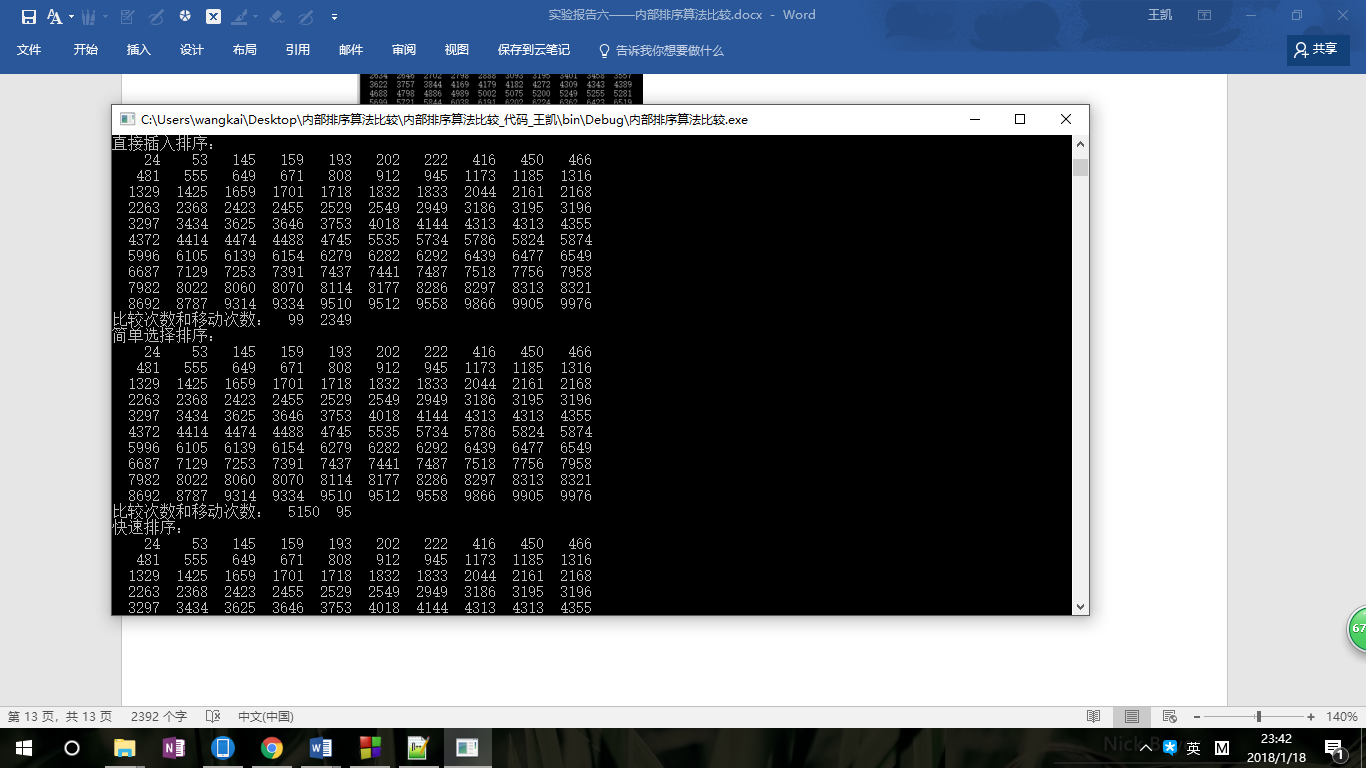


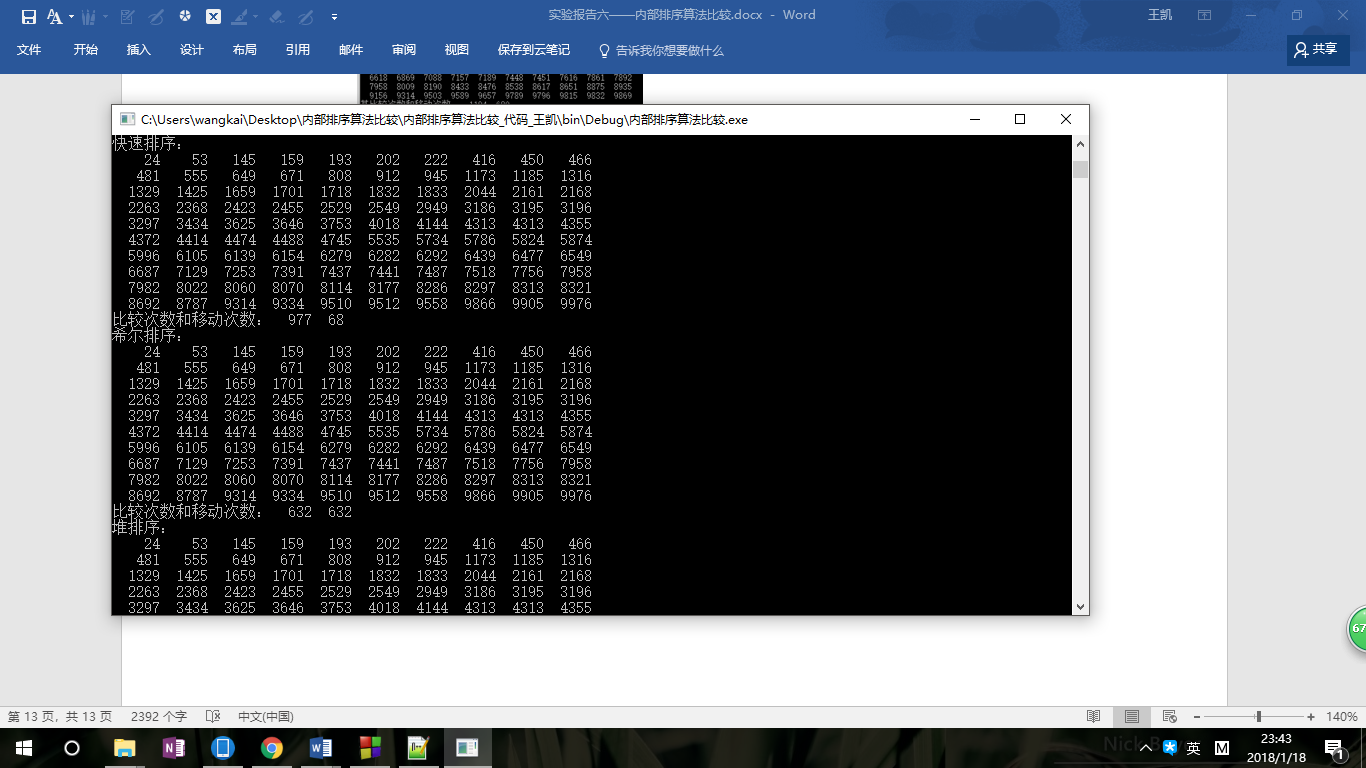
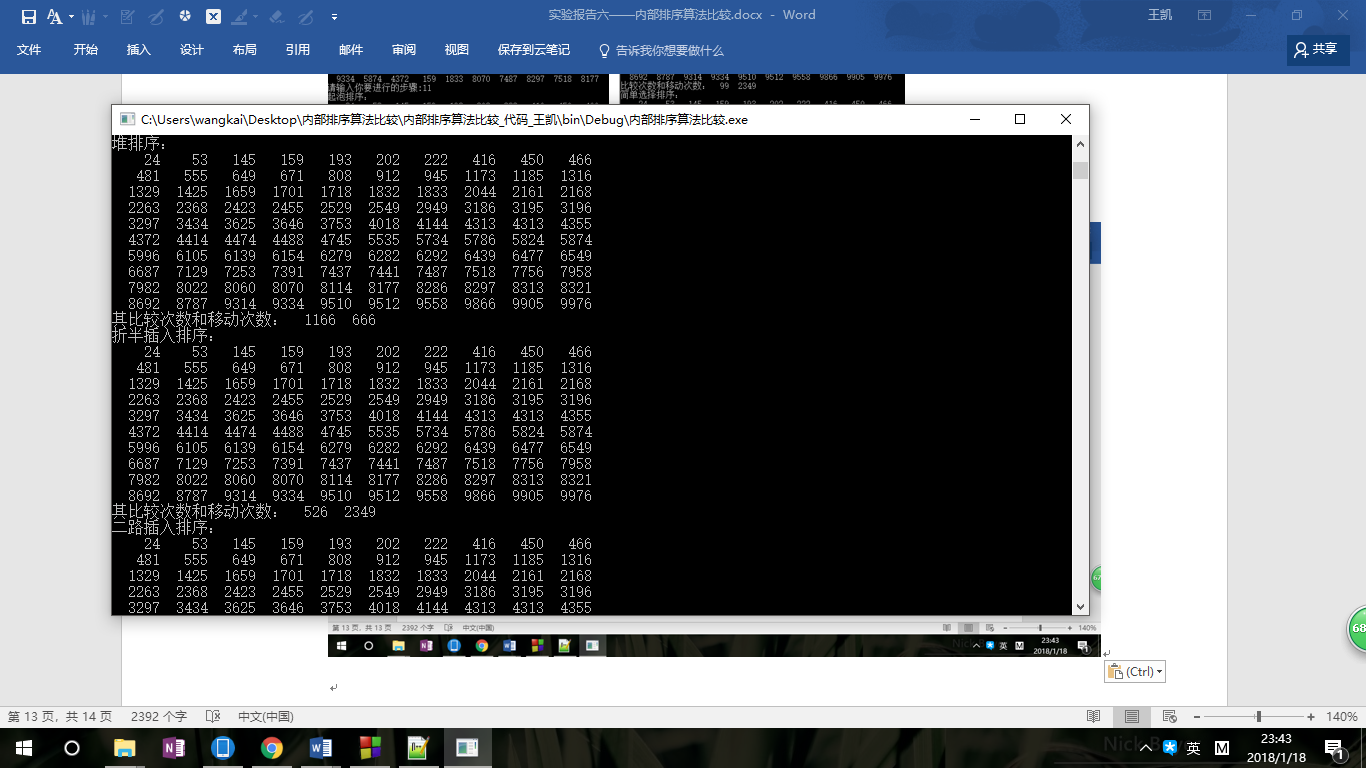
6：堆排序

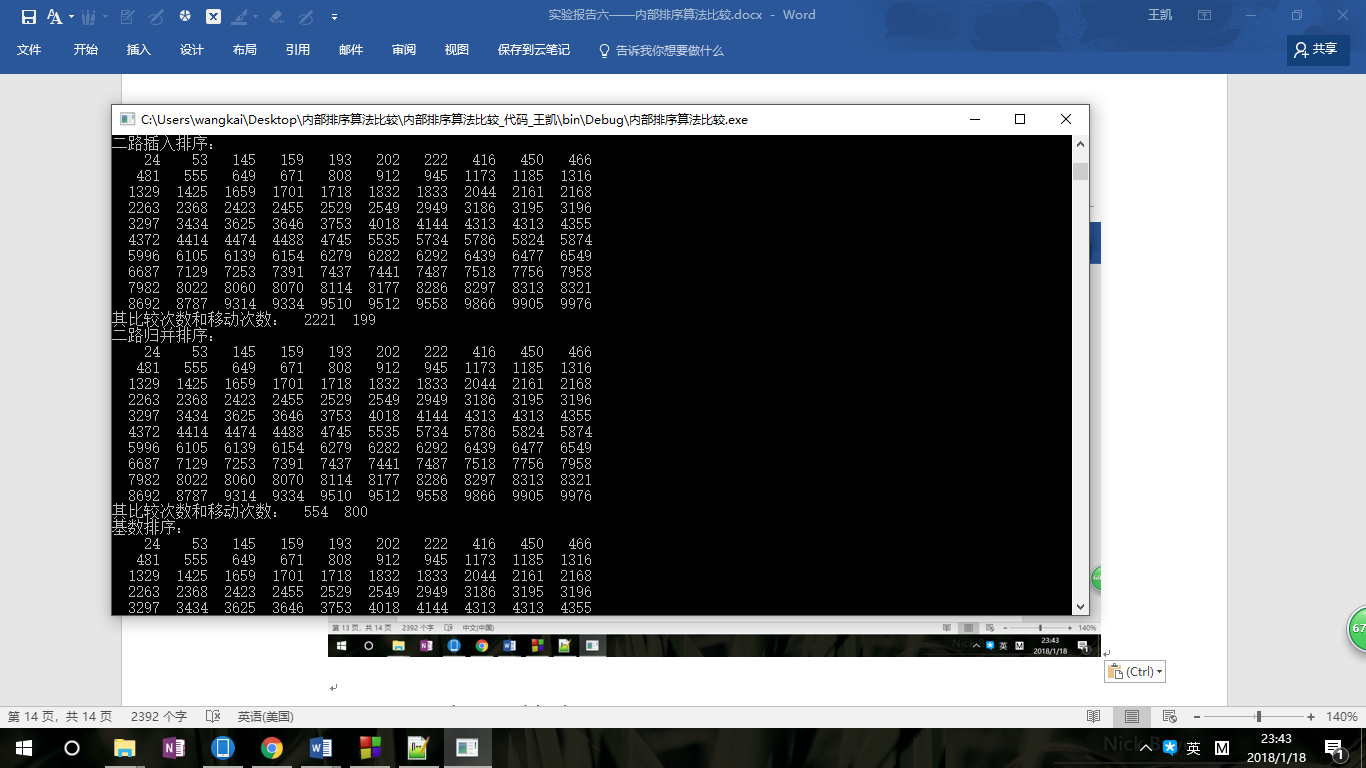


另外集中排序的结果不在赘述。

下面看一下对于一组数据，10种排序的结果：



、

附录：源码见文件 稀疏矩阵运算器”